Give Psuedocode for algorithm: Exercise R10.3. Outline, but do not implement, a recursive solution for finding the smallest value in an array.

A function that takes an array and an int(zero when first called) and an element from the array passed.

Check if the int passed as a parameter is the number of elements in the array.

If it is the number of elements in the array than the element passed to the function is returned as the smallest value in the array.

If the int is below the number of elements in the array, it checks if the element at the index of the int passed to the function is less than the element passed to the function.

If it is less than, then the function is called, passing the array, the int + 1, and the element originally passed to it. Otherwise, the function is called passing the array, the int + 1, and the element the original element was compared to.

Give Pseudocode for algorithm: Exercise R10.7. Write a recursive definition of xn, where x ≥ 0, similar to the recursive definition of the Fibonacci numbers. Hint: How do you compute xn from xn–1? How does the recursion terminate? (first i dont know what you mean by xn? Is it suppose to be x^n? As that would make more sense and is what im going to assume for the assignment. I know you all didn't write this book but it would have been helpful if you included a note next to the assignment as you did saying "give pseudocode for the algorithm")

Function returning an int, taking two ints (x and n). Create a int variable named m, If n is equal to zero return 1, if n is divisible by 2, then m is equal to the function passed x and n divided by 2 and then return m * m. If n is not divisible by 2 then return x multiplied by the function passed x and n minus 1.

Submit Function definition in CPP (Selection Sort is described and implemented in Section 11.1):
Exercise R11.1. Checking against off-by-one errors. When writing the selection sort algorithm of Section 11.1, a programmer must make the usual choices of < against <=, a.size() against a.size() - 1, and next against next + 1. This is fertile ground for off-by-one errors. Make code walkthroughs of the algorithm with vectors of length 0, 1, 2, and 3 and check carefully that all index values are correct.

```
int min_position(vector& a, int from, int to)
{
int min_pos = from;
int i;
for (i = from + 1; i <= to; i++)
        if (a[i] < a[min_pos]) min_pos = i;
                return min_pos;
}

void selection_sort(vector& a)
{
        int next;
        for (next = 0; next < a.size() - 1; next++)
        {
                int min_pos = min_position(a, next, a.size() - 1);
                if (min_pos != next)
                        swap(a[min_pos], a[next]);
        }
}

int main()
{
        rand_seed();
        vector v(//case for the walkthrough);
        for (int i = 0; i < v.size(); i++)
                v[i] = rand_int(1, 100);
        print(v);
        selection_sort(v);
        print(v);
        return 0;
}
```

Case 0:
No random numbers would be assigned to the vector
Nothing would be printed
Nothing would be sorted
Nothing would be printed again

Case 1:
The only element would be assigned a random value
The value would be printed out to the user
The selection Sort would do nothing
The value would be printed again to the user

Case 2:
Random numbers would be assigned to the two elements in the vector
The numbers would be printed to the user
The vector would be sent to the selection sort method
If the two values were not equal they would be swapped by the swap function
The resulting sort would be printed to the user

Case 3:
Random numbers assigned to the elements of the vector
The numbers would be printed to the user
The vector would be sent to the selection sort method
Selection sort would check if the first and the next values in the vector are equal, and based on that would swap the values or not, and do the same for the second and last values in the vector.
The sorted vector would be printed to the user.

Give CPP code: Exercise R7.2. A pointer variable can contain a pointer to a valid object, a pointer to a deleted object, NULL, or a random value. Write code that creates and sets four pointer variables a, b, c, and d to show each of these possibilities.

```
Int * p = 0; // null

p = &myObject; //object

int* p;
p = new int;     //deleted object
 *p = 6;

int* q2 ; // random pointer
```

(You can read a bit about the heap throughout Chapter 7): Exercise R7.4. What happens if you forget to delete an object that you obtained from the heap? What happens if you delete it twice?

A memory leak can occur, if you delete it twice it will corrupt the heap.